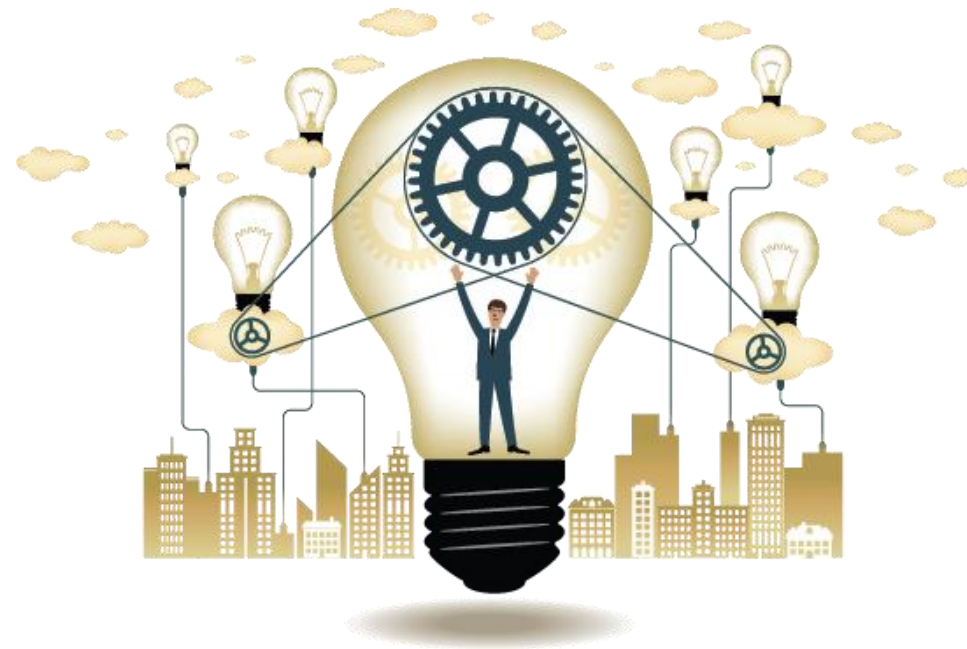


EABSS Workshop 2023

Co-Creation of Agent-Based Social Simulation Models

UML (Unified Modelling Language)

Unified Modelling Language

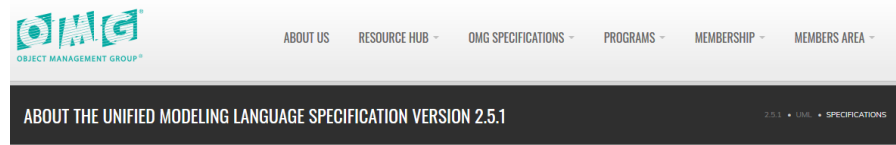


UML: An Overview

- UML: "A specification defining a graphical language for visualizing, specifying, constructing, and documenting the artifacts of distributed object systems."

<https://www.omg.org/spec/UML/About-UML/>

- Latest Version: 2.5.1 (Dec 2017)

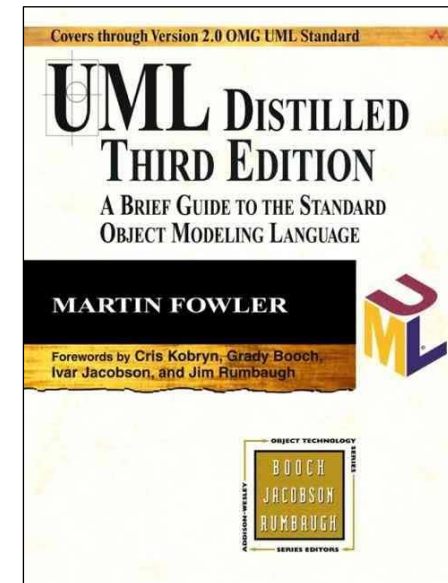


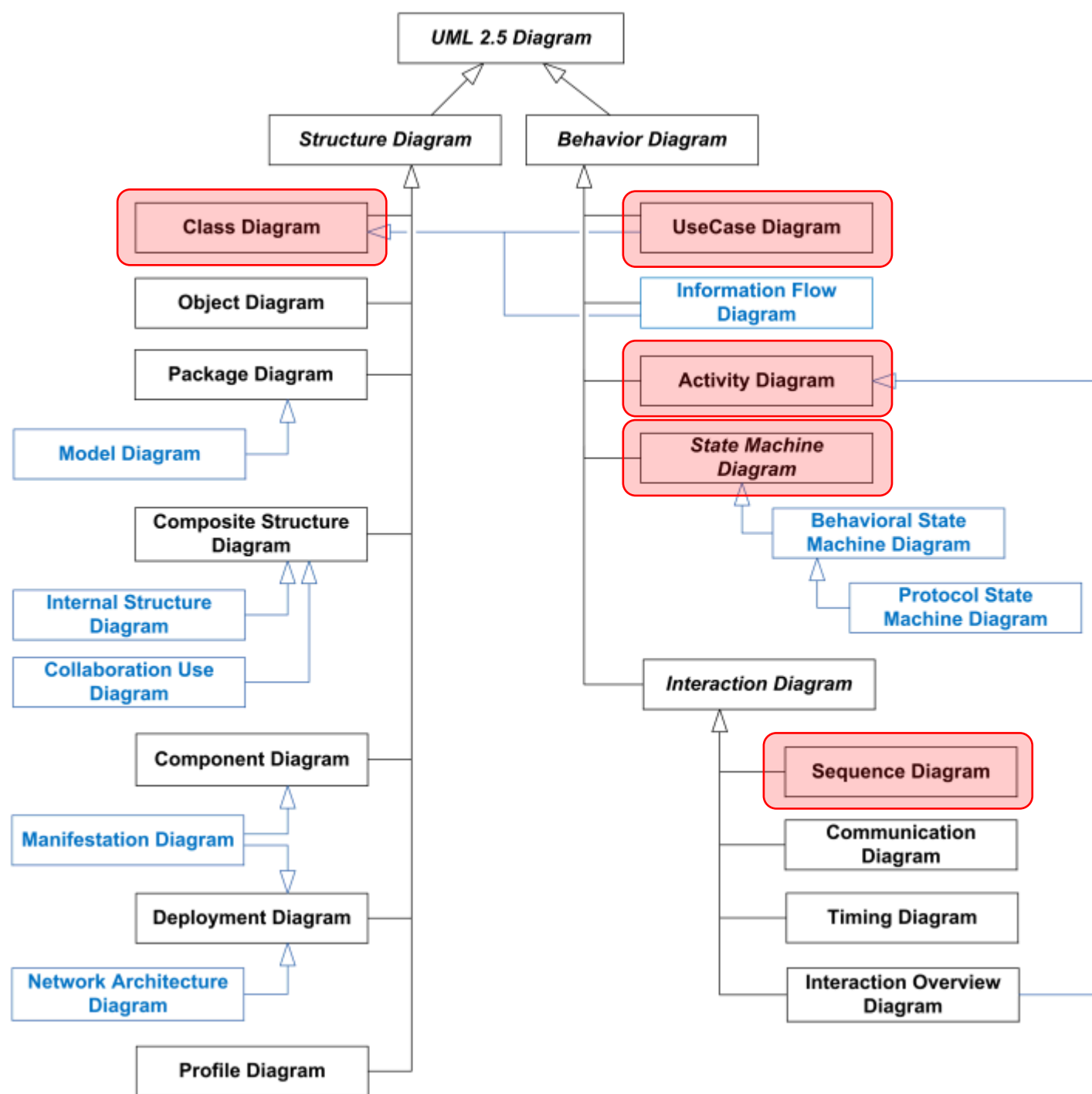
UML® Unified Modeling Language

A specification defining a graphical language for visualizing, specifying, constructing, and documenting the artifacts of distributed object systems.

Title:	Unified Modeling Language
Acronym:	UML®
Version:	2.5.1
Document Status:	formal
Publication Date:	December 2017
Categories:	Modeling Software Engineering
IPR Mode:	RF-Limited

Specification







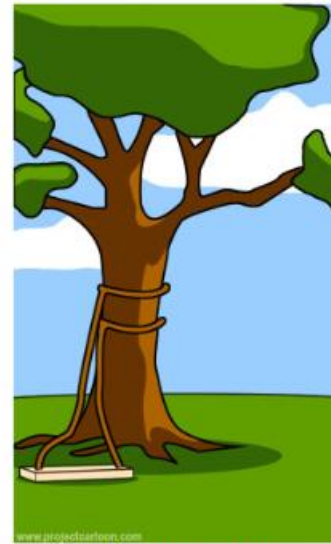
How the customer explained it



How the project leader understood it



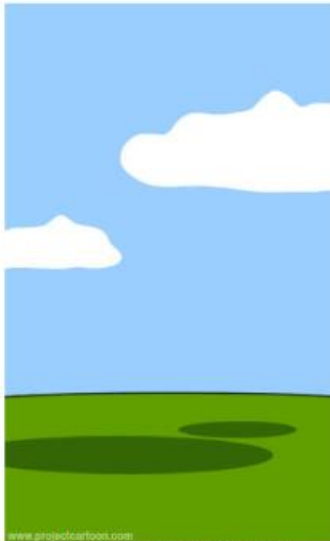
How the analyst designed it



How the programmer wrote it



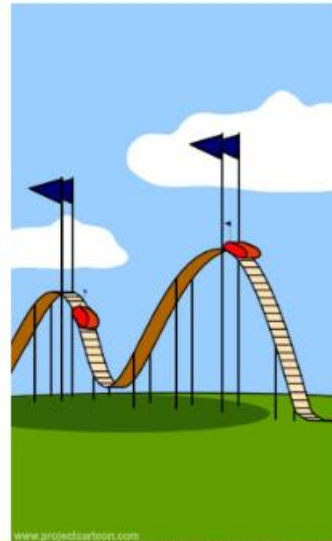
How the business consultant described it



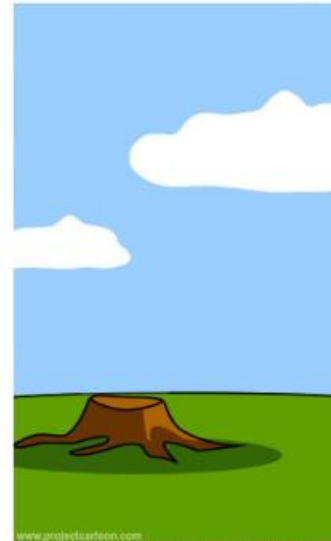
How the project was documented



What operations installed



How the customer was billed



How it was supported



What the customer really needed

Why UML?

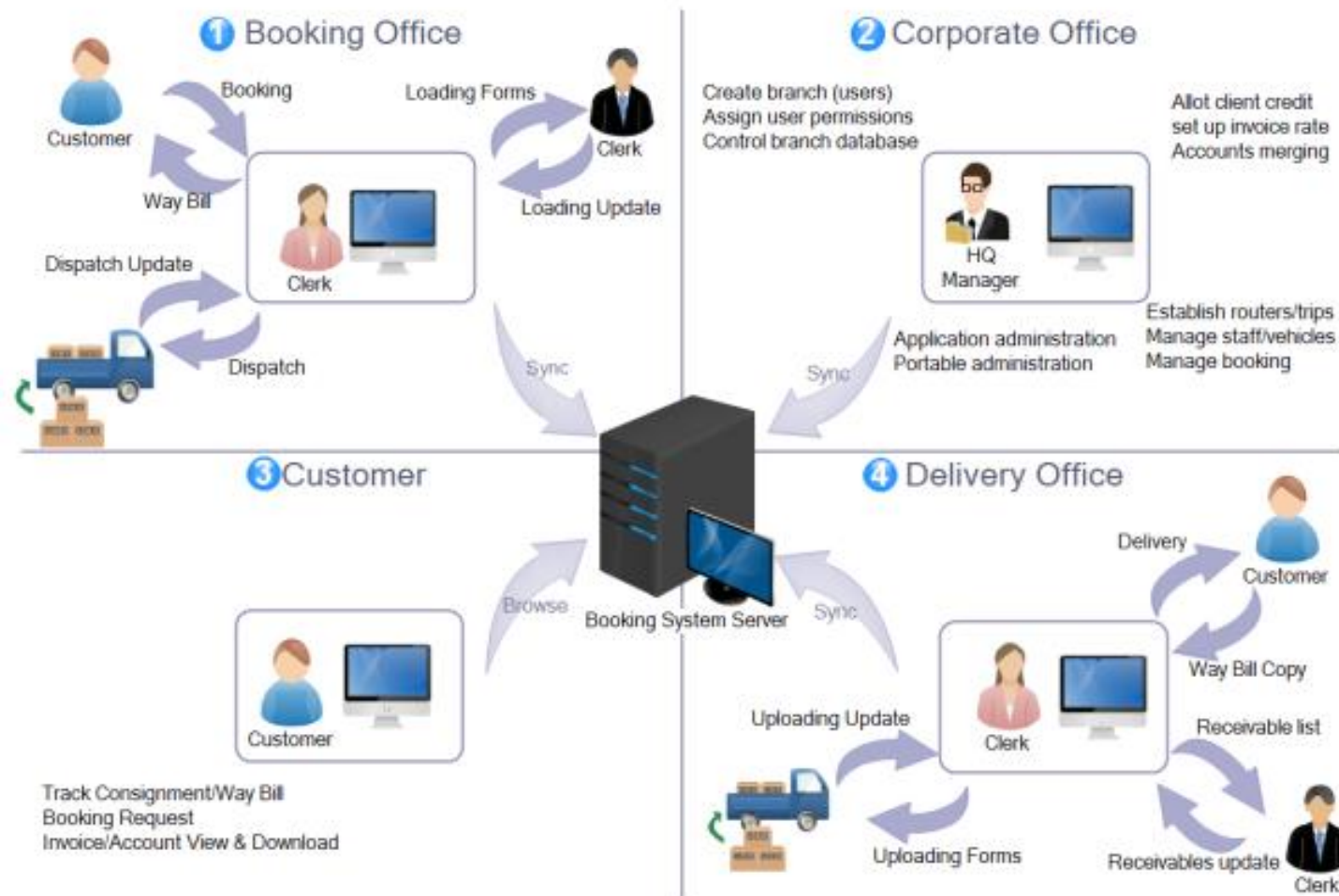
- Advantages of using UML:
 - Enhances communication and ensures the right communication
 - Captures the logical software architecture independent of the implementation language
 - Helps to manage the complexity
 - Enables reuse of design

Case Study





Case Study: Fleet Logistics Management

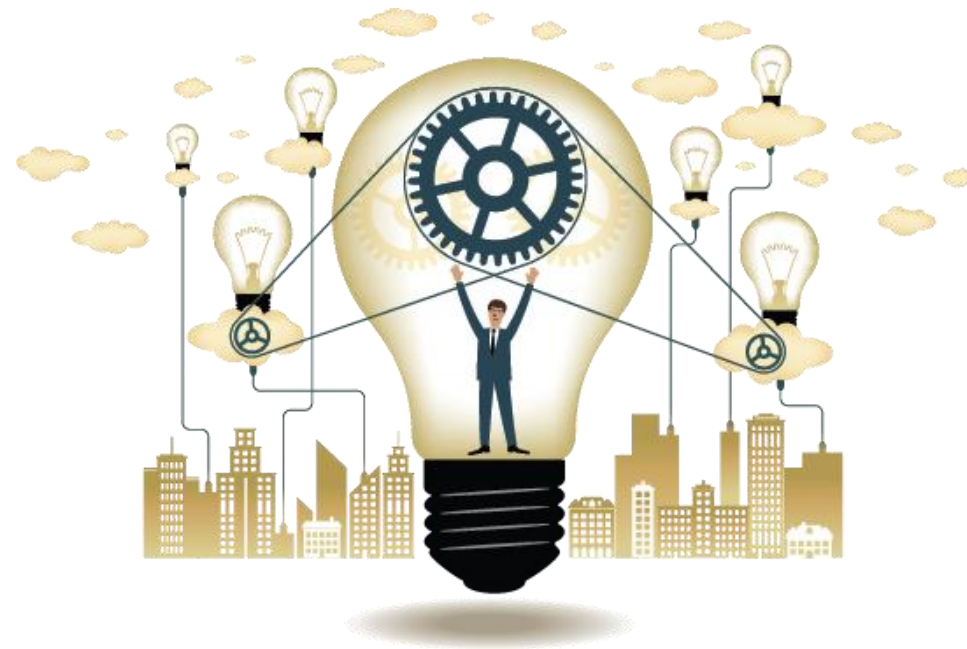




Case Study: Fleet Logistics Management

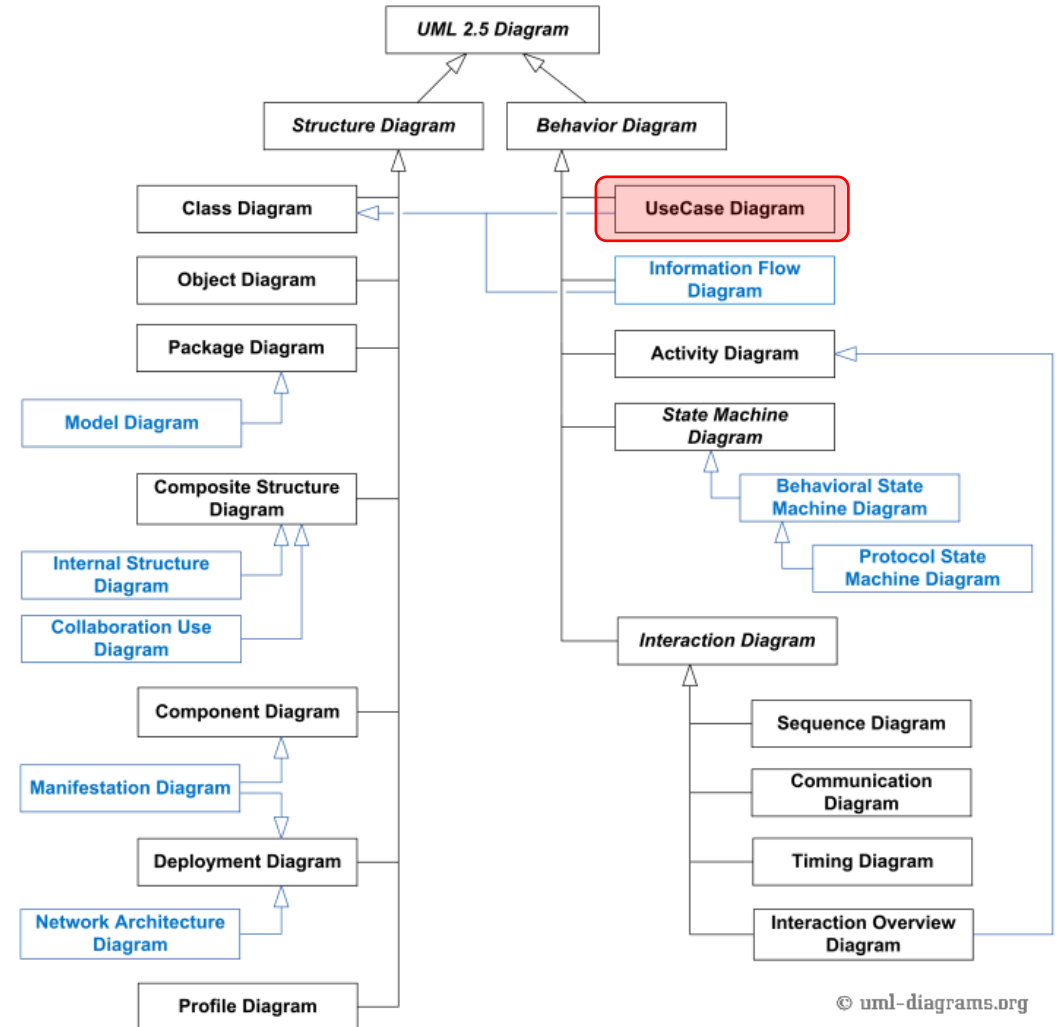
- User stories
 - As a client I want to be able to check availability of lorries
 - As a client I want to be able to track cargo
 - As a manager I want to be able to see the finances
 - As an admin I want to be able to search for information
 - As an admin I want to be able to organise routes
 - As an admin I want to be able to track lorries and cargo

Object Oriented Analysis



Components Covered

- We will be looking at:
 - Use case diagrams
 - Use case specifications



© uml-diagrams.org

Use Case Diagrams

- What are use case diagrams used for?
 - They describe a set of actions that some system or systems should or can perform in collaboration with one or more external users of the system or systems
 - No attempt to represent an order or a number of executions
- Use case diagram components
 - Actors
 - Use cases
 - Subject (also referred to as "system boundary")
 - Relationships

Use Case Diagrams

- Actors
 - Entities that interface with the system
 - Can be people or other systems
- Use cases
 - Based on user stories
 - Represent what the actor wants your system to do for them
 - In the use case diagram only the use case name is represented
 - In a use case specification each use case is formally described

Use Case Diagrams

- **Subject** (also referred to as "system boundary")
 - Classifier representing a business, software system, physical system or device under analysis, design, or consideration
- **Relationships**
 - Relationship between use case and actor:
 - Association indicates which actor **initiates** which use case
 - Relationship between two use cases:
 - Specifying common functionality and simplifying use case flows
 - Using <<include>> or <<extend>>

Use Case Diagrams

- <<include>>
 - Used when multiple use cases share a piece of same functionality which is placed in a separate use case
 - Arrow points to the more specific use case
- <<extends>>
 - Used when activities might be performed as part of another activity but are not mandatory for a use case to run successfully
 - Arrow points to the more general use case

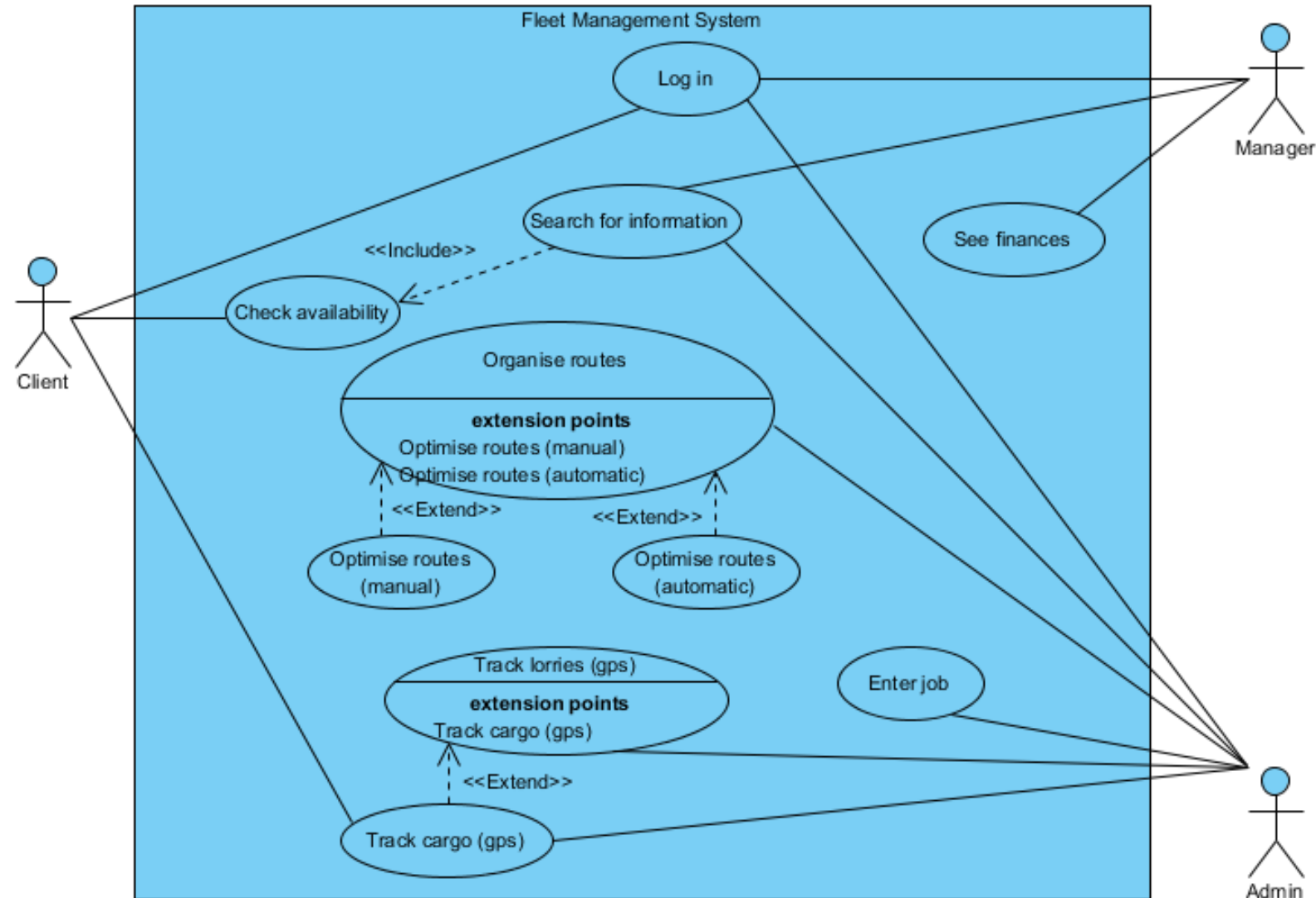


Case Study: Fleet Logistics Management

- Use Case Diagram
 - Reminder
 - Client wants to check availability of lorries and track cargo.
 - Manager wants to see the finances.
 - Admin wants to search for information, organise routes and track lorries and cargo



Case Study: Fleet Logistics Management

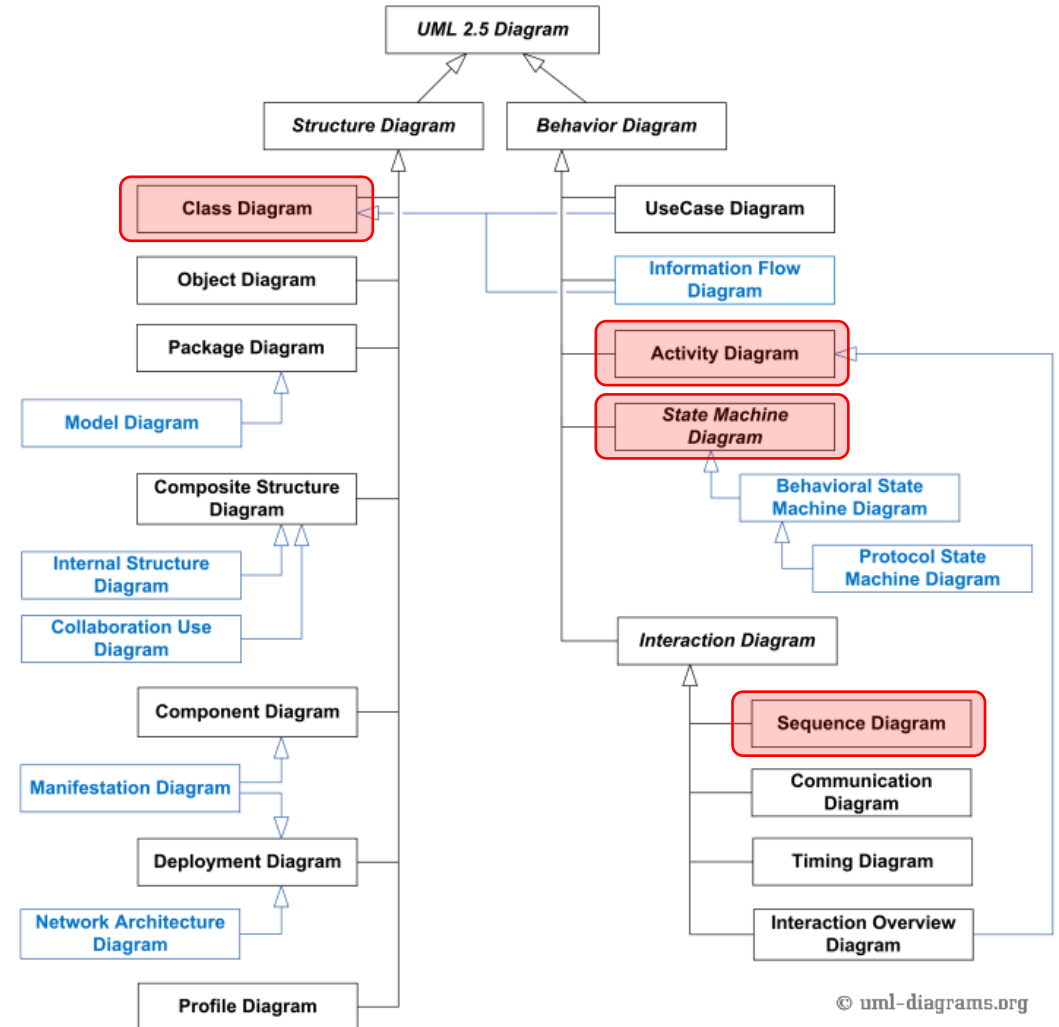


Object Oriented Design



Components Covered

- We will be looking at:
 - Activity diagrams
 - Sequence diagrams
 - State machine diagrams
 - Class diagrams



Activity Diagrams

- What are activity diagrams used for?
 - Graphical representations of workflows of stepwise activities and actions related to an individual use case or across many use cases
 - Support representation of parallel behaviour
- Activity diagram components
 - Activity: A state that is left once the activity is finished
 - Activity edge: Transition that fires when previous activity completes
 - Synchronisation bar: Describes the co-ordination of activities
 - Decision diamond: Used to show decisions
 - Start and stop marker: Used to define entry and exit points
 - Swim lane: A way to group activities performed by the same actor on an activity diagram or to group activities in a single thread

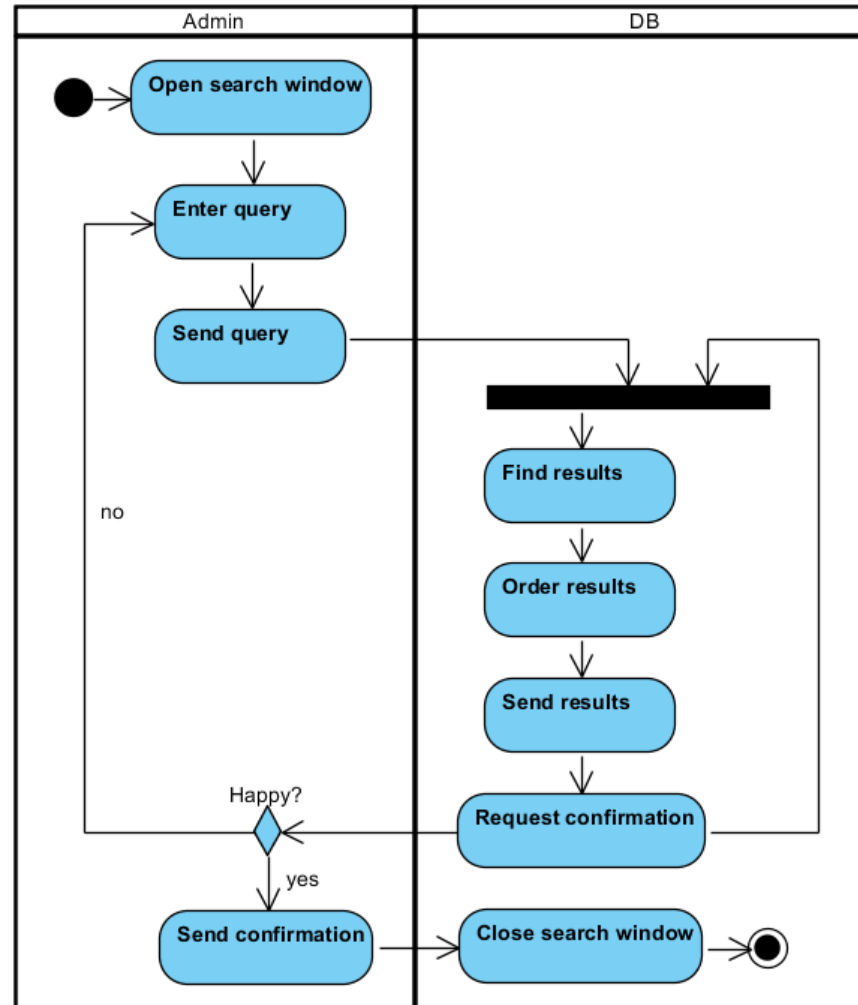


Case Study: Fleet Logistics Management

- Activity Diagram for use case "Search for Information"
 - Reminder: Base Path (optimistic flow) for this use case
 1. Admin opens search window
 2. Admin defines query using query editor
 3. Admin sends query to DB
 4. DB deals with query: finding results
 5. DB deals with query: organising them by relevance
 6. DB sends results back
 7. DB requests confirmation that result is sufficient
 8. Admin confirms that result is sufficient
 9. DB closes search window



Case Study: Fleet Logistics Management



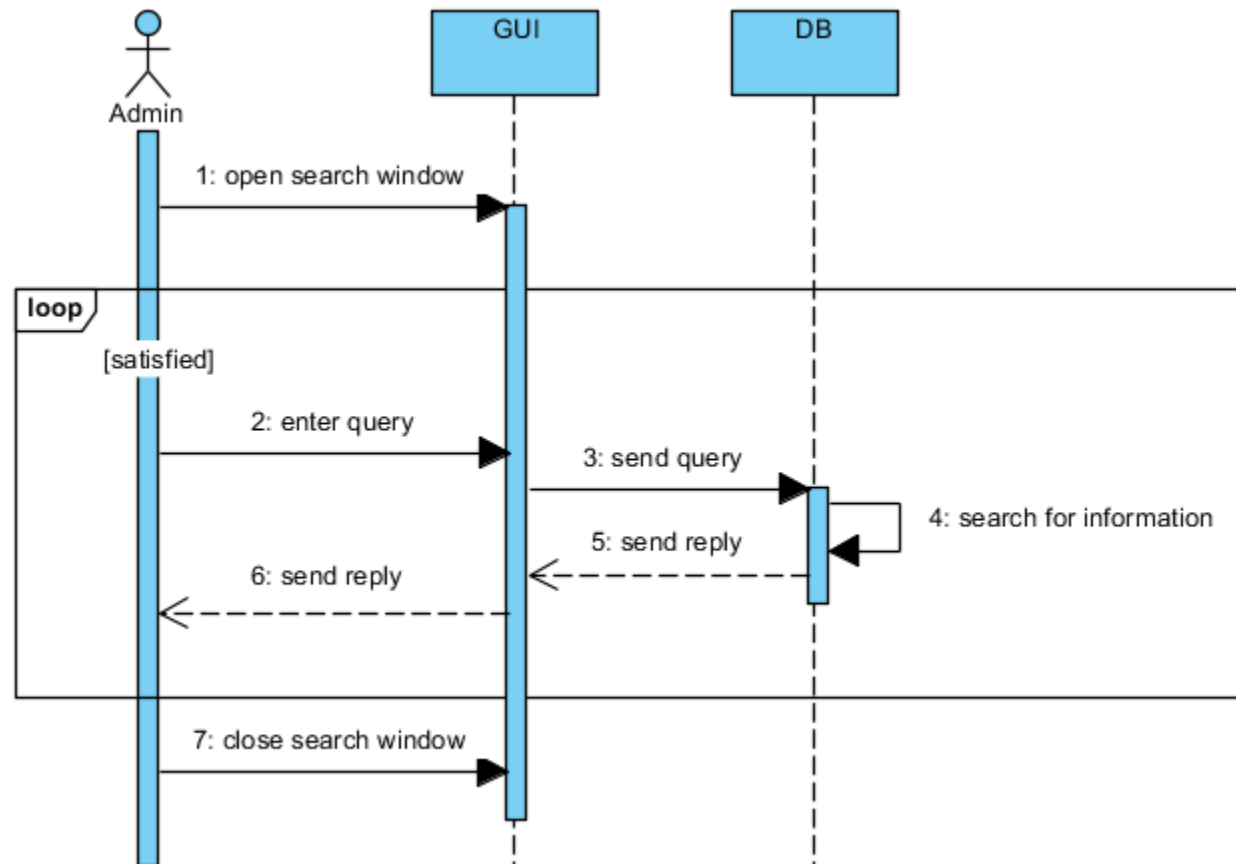
Sequence Diagrams

- What are sequence diagram used for?
 - Sequence diagrams are a temporal representation of objects and their interactions
- Sequence diagram components
 - Participant: Object or actors that act in the sequence diagram
 - Vertical line (called lifeline): Represent time as seen by the object
 - Narrow rectangle covering an object's life line shows a live activation of the object
 - Arrow from sender's lifeline to receiver's lifeline:
 - General: Message, denoting an event or the invocation of an operation
 - Object creation: Arrow with 'new' written above it
 - Object deletion: An X at bottom of lifeline (in some OOP languages this is handled automatically)
 - Sequence fragment: Let you show loops, branches, and other alternatives



Case Study: Fleet Logistics Management

- Sequence Diagram for use case "Search for Information"



State Machine Diagrams

- What are state machine diagrams used for?
 - Show the possible states of a single object, the events or messages that cause a transition from one state to another, and the action that result from that state change
 - Only reactive objects require a state machine diagram!
- State machine diagram components
 - State: A condition during the life of an object when it satisfies some condition, performs some action, or waits for an event. Special states:
 - Start state: Each state diagram must have one and only one start state
 - Stop state: An object can have multiple stop states

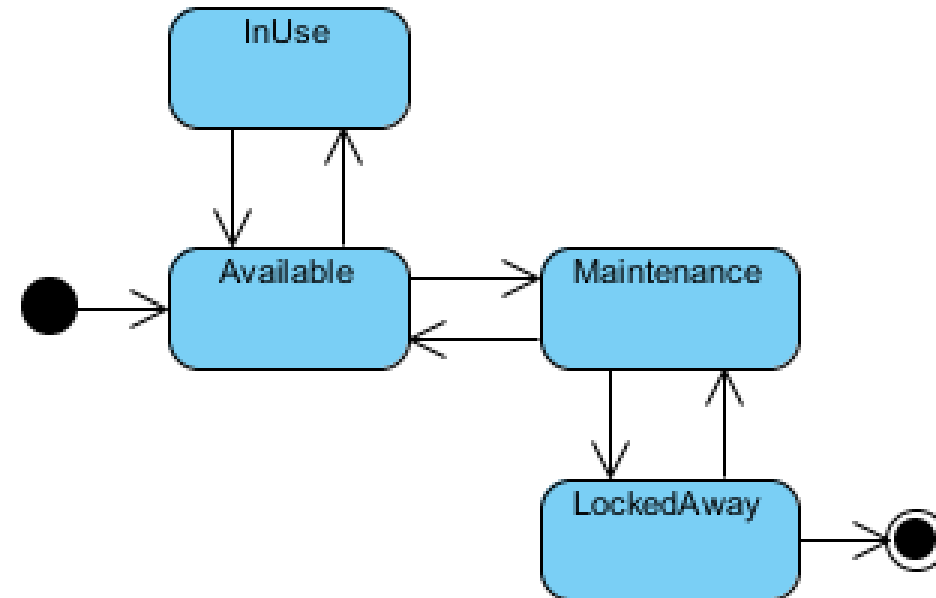
Case Study: Fleet Logistics Management

- State machine diagram components (continued)
 - Transition: Denotes the transition between states or to the same state (self-transition)
 - A transition may have a trigger, a guard and an effect (Trigger[Guard]/Effect):
 - Trigger: The cause of the transition (signal; event; change in some condition; passage of time)
 - Guard: Condition which must be true in order for the trigger to cause the transition
 - Effect: Action invoked directly on the object that owns the state machine as a result of the transition



Case Study: Fleet Logistics Management

- State Machine Diagram for "Lorry" Class



Class Diagrams

- What is a class diagram used for?
 - A class diagrams shows the existence of classes and their structures and relationships in the logical view of a system
- Class diagram's main components:
 - Classes
 - Class relationships
 - Associations; dependencies; aggregations; compositions; realisations; generalisations
 - Multiplicity indicators

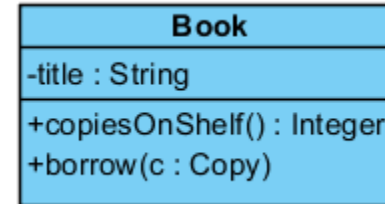
Class Diagrams

- Class representation

- In UML classes are depicted as rectangles with three compartments
 - Class name
 - Attributes: Describe the data contained in an object of the class
 - Operations: Define the ways in which objects interact

- Additional symbols

+	public
#	protected
-	private
/	derived
\$	static



This is the record that keeps track of the books

Class Diagrams

Associations between classes

- Classes are associated if an instance of class A (the source class) has to know about an instance of class B (the target class) or vice versa

Multiplicity indicators

- Number of links between each instance of the source class and instances of the target class
 - 1 = exactly 1
 - * = unlimited number (zero or more)
 - 0..* = zero or more
 - 1..* = one or more
 - 0..1 = zero or 1
 - 3..7 = specified range (from 3 to 7)

Class Diagrams

- Relationship: Association
 - Reference based relationship between two classes
 - Class A holds a class level reference to class B
 - Represented by a line between A and B with an arrow indicating the navigation direction
 - If no arrow or arrow on the both sides, association has bidirectional navigation



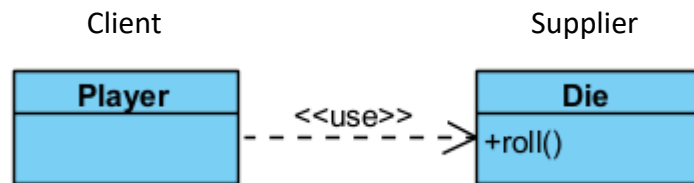
```
1 class Asset {
2     ...
3 }
4
5 class Player {
6     private Asset asset
7     ...
8     public Player(Asset purchasedAsset) {
9         this.asset = purchasedAsset;
10        ...
11    }
12 }
```

Class Diagrams

- Relationship: Dependency

- Created when you receive a reference to a class as part of a particular method
- Dependency indicates that you may invoke one of the APIs of the received class reference and any modification to that class may break your class as well
- Multiplicity does not make sense on a Dependency

<<use>> The source (Player) requires the target (Die) for the implementation

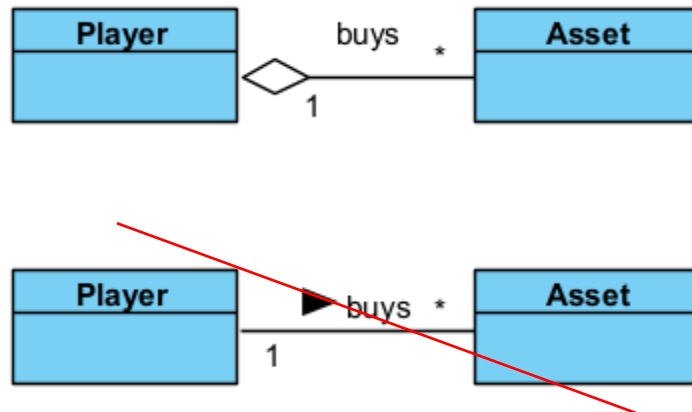


Supplier-client relationship, where the supplier provides something to the client, and thus the client is in some sense incomplete while semantically or structurally dependent on the supplier element(s). Modification of the supplier may impact the client elements.

```
1 class Die {
2     ...
3     public void roll() {...}
4 }
5
6 class Player {
7     ...
8     public void takeTurn(Die die) {
9         die.roll();
10        ...
11    }
12 }
```


Class Diagrams

- Relationship: Aggregation ("is part of" relationship)
 - Often seen as redundant relationship as technically the same as an association; but semantically there is a difference
 - It is used when an object logically or physically contains another; the container is called "aggregate"; the components of the aggregate can be shared with others

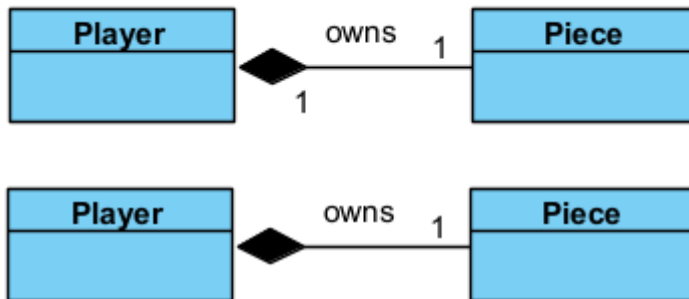


```
1 class Asset {  
2     ...  
3 }  
4  
5 class Player {  
6     private List assets  
7     ...  
8     public void addAsset(Asset purchasedAsset) {  
9         ..  
10        assets.add(purchasedAsset);  
11        ...  
12    }  
13 }
```

Class Diagrams

- Relationship: Composition

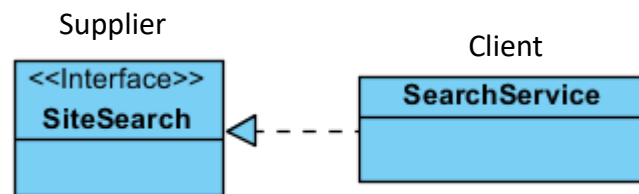
- Relates to instance creational responsibility
- When class B is composed by class A, class A instance owns the creation or controls lifetime of instance of class B
 - Composition binds lifetime of a specific instance for a given class, while class itself may be accessible by other parts of the system.
- Multiplicity at the composition end is always 1 as parts have no meaning outside the whole



```
1 class Piece {  
2     ...  
3 }  
4  
5 class Player {  
6     private Piece piece=new Piece();  
7     ...  
8 }
```

Class Diagrams

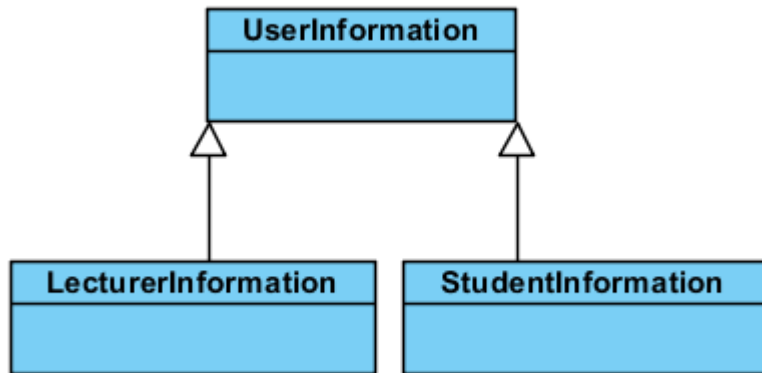
- Relationship: Realisation
 - A "Realisation" is a specialised abstraction relationship between two sets of model elements, one representing a specification (the supplier) and the other representing an implementation (the client) of the specification



```
1 class SearchService implements SiteSearch {  
2     ...  
3 }
```

Class Diagrams

- Relationship: Generalisation ("is a" relationship)
 - A directed relationship between a more general classifier (superclass) and a more specific classifier (subclass)



```
1 class LecturerInformation extends UserInformation {
2     ...
3 }
4
5 class StudentInformation extends UserInformation {
6     ...
7 }
```

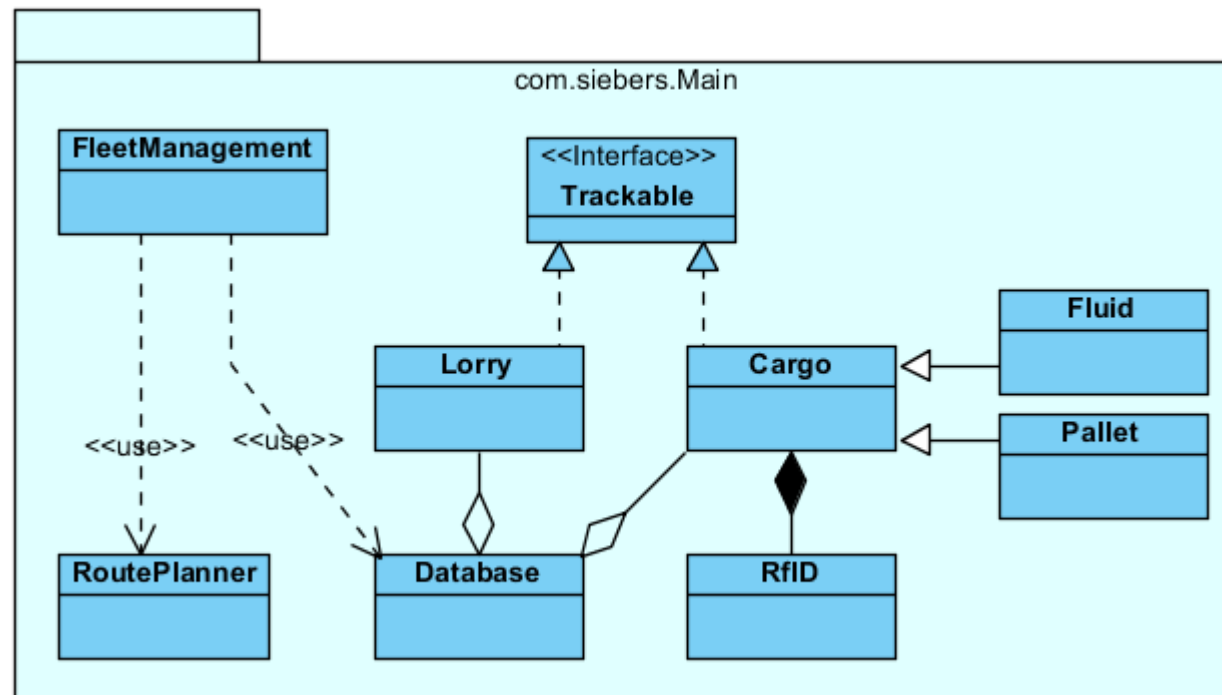


Case Study: Fleet Logistics Management

- Class Diagram
 - Reminder
 - Client wants to check availability of lorries and track cargo. Manager wants to see the finances. Admin wants to search for information, organise routes and track lorries and cargo



Case Study: Fleet Logistics Management



Any Questions?



References

- Barclay and Savage (2004) Object-Oriented Design with UML and Java
- Some of the relationship descriptions taken from <https://nirajrules.wordpress.com/2011/07/15/association-vs-dependency-vs-aggregation-vs-composition/>